Performance of A Supernodal General Sparse Solver on the CRAY Y-MP: 1.68 GFLOPS with Autotasking

Horst D. Simon, [1] Phuong Vu, and Chao Yang [2]

Report RNR-89-002, April 1989

NAS Systems Division
NASA Ames Research Center, Mail Stop 258-5
Moffett Field, CA 94035

April 6, 1989

[1] The first author is an employee of Boeing Computer Services, Bellevuew, WA 98128

[2] The second and third authors are employees of Cray Research Inc., Mendota Heights, MN 55120

Sparse Matrix Factorization at 1.68 GFLOPS

Horst D. Simon,[1]
Numerical Aerodynamic Simulation (NAS) Systems Division
NASA Ames Research Center, Mail Stop 258-5
Moffett Field, CA 94035

Phuong Vu,
and
Chao Yang
Cray Research Inc.
1345 Northland Drive
Mendota Heights, MN 55120

February 24, 1989

Abstract

Numerical results with a general sparse linear equation solver package are presented. This software has been developed for production use by researchers at Boeing Computer Services and is available from Cray Research for Cray X-MP systems. Recently this package has been ported to the new eight processor Cray Y-MP. Initial results using microtasking and autotasking primitives and all eight processors yielded 1682 MFLOPS for the factorization of a sparse matrix of order 24,565. The results reported here demonstrate the high performance levels obtainable on the new Cray Y-MP in autotasking mode. We also believe that this is the highest speed ever obtained factoring a non-trivial sparse matrix.

[1]The author is an employee of Boeing Computer Services.

1 Introduction

The efficient solution of general sparse linear systems of equations is a task common to a variety of important engineering and scientific applications, such as structural analysis, computational fluid dynamics, economic modeling, chemical engineering, circuit and device simulation, and electric power network problems [7]. From an algorithmic perspective the implementation of general sparse methods on vector and parallel machines poses a great intellectual challenge. Over the last couple of years significant progress has been made in implementing efficient sparse matrix methods on these machines. The multifrontal method, which is an alternative to the approach described below, has been developed by Duff and Reid [4, 5]. It has been implemented on the Cray X-MP by Ashcraft [3], and on the Cray 2 by Amestoy and Duff [1]. General sparse methods for distributed memory parallel machines, in particular hypercubes have been implemented by George et al. at Oak Ridge (see [8] for a summary and survey).

We were involved in the development of a production software package for the efficient solution of these sparse systems [2], using the new algorithmic concept of a supernodal sparse factorization. Very high speeds have been obtained in single processor mode on a Cray X-MP as documented in [2, 11]. Recently this code has been ported to the eight processor Cray Y-MP, and modified for parallel execution using the microtasking and autotasking primitives.

The Cray Y-MP is a shared memory machine with a relatively small number of powerful vector processors. The Cray Y-MP has a cycle time (depending on the model) of 6.0 to 6.49 nanoseconds. The achievable peak for a single processor Y-MP with a 6.0 ns clock is 313.7 MFLOPS.

A good overall strategy for achieving high performance on a machine like the Y-MP is obtain a high degree of vectorization first, and then to utilize multiple processors without destroying vectorization. On a machine like the Y-MP more substantial gains can be made with good vectorization (speed-ups by a factor of 20 over scalar code), than with good parallelization (speed-ups by a factor of 8 over single processor performance).

After autotasking a highly vectorized code we were able to obtain speeds of up to 1.68 GFLOPS. We believe that this is currently the fastest production quality sparse linear solver available.

2 Vectorization

Let us first describe our strategy to obtain a high degree of vectorization. Until recently general sparse matrix methods have not gained wide acceptance among the developers of software for large scale applications. This was chiefly due to the indirect addressing required by the numerical data structures. Hence these codes did not vectorize on first generation supercomputers [10]. This situation was improved with the availability of hardware gather-scatter on most recent vector computers.

However, the major improvement was the application of the idea of a supernode. Similar to dense linear algebra operations, the key to high performance for sparse methods is to recognize where double or triple-level-loops can be vectorized by loop unrolling or related techniques. The use of vectorization at a higher level results in a reduction of memory traffic. In the sparse case the memory references saved are gather/scatter operations, which are usually more costly than their dense counterparts.

The key ideas come from the graph theory model of the sparse elimination process [9]. It has been observed that successive steps of elimination generate a coalescing of sparsity structure in the matrix. This means that several columns in the factored matrix share essentially the same sparsity pattern. In the graph theory model the nodes of the graph corresponding to columns with the same sparsity structure are said to be indistinguishable. We call each set of indistinguishable nodes a supernode. The shared sparsity pattern of a supernode allows us to carry out the elimination as a sequence of supernodal updates. Computationally that means that we have replaced the sequence of gather-SAXPY-scatter by gather-multiple SAXPY-scatter. Thus a reduction of memory references results. Further speed-up is obtained by implementing the multiple SAXPY's as a matrix vector multiply, which can be sped up using standard loop-unrolling techniques. More details on this approach and numerical results on the Cray X-MP are given in [2].

3 Microtasking and Autotasking

Microtasking and autotasking are some of the tools which are provided on the Cray Y-MP to utilize multiple processors. Both are based on directives. With these directives a user can indicate, for example, that the iterations of

an outer loop can be executed in arbitrary order, and thus can be distributed over multiple processors. Autotasking is the more recent product and in many ways an improvement over microtasking. Autotasking directives are more flexible and are implemented more efficiently.

Using profiling tools it was identified that the majority of the execution time for the sparse matrix factorization was spent in two matrix-vector multiplication subroutines. These two routines are the result of the supernode operations described above, and involve one or two levels of indexed addressing of the matrix data. These routines were microtasked, and more recently autotasked by breaking the matrix into p submatrices, consisting of n/p rows. Here p is the number of processors available.

The autotasking implementation is very sophisticated about when to use the autotasked version of the kernels and when to use the single processor version. The switch-over points have been determined very carefully. On the factorization with the highest speed, we observed a performance deterioration by about 200 MFLOPS, when the autotasked subroutine was used throughout the factorization phase. This determination of the optimal switch-over points was the main difficulty in obtaining a high performance parallel implementation. The actual mechanics of autotasking were relatively simple.

Figure 1 shows schematically the interplay of parallelization and vectorization, with "strip-mining" the doubly-indexed matrix-vector multiply, without destroying the carefully tuned vectorization. Note that Figure 1 is a conceptual simplification, since the underlying matrix is not contiguously stored in memory.

4 Performance Results

The numerical results with the vectorized supernodal code have appeared already in the literature [2]. The results in [2] demonstrate a factorization performance in excess of 100 MFLOPS on the 9.5 nanosec Cray X-MP using all FORTRAN code, and loop unrolling to a depth of eight. These results were obtained on real structural analysis problems from the Boeing-Harwell Sparse Matrix Test Collection [6]. The models analyzed include the rear bulkhead of a Boeing 767 aircraft (BCSSTK29) and a 76 story high-rise building in downtown Seattle (BCSSTK25). In the meantime these numbers have been improved for the X-MP. Using assembly coded subroutines one

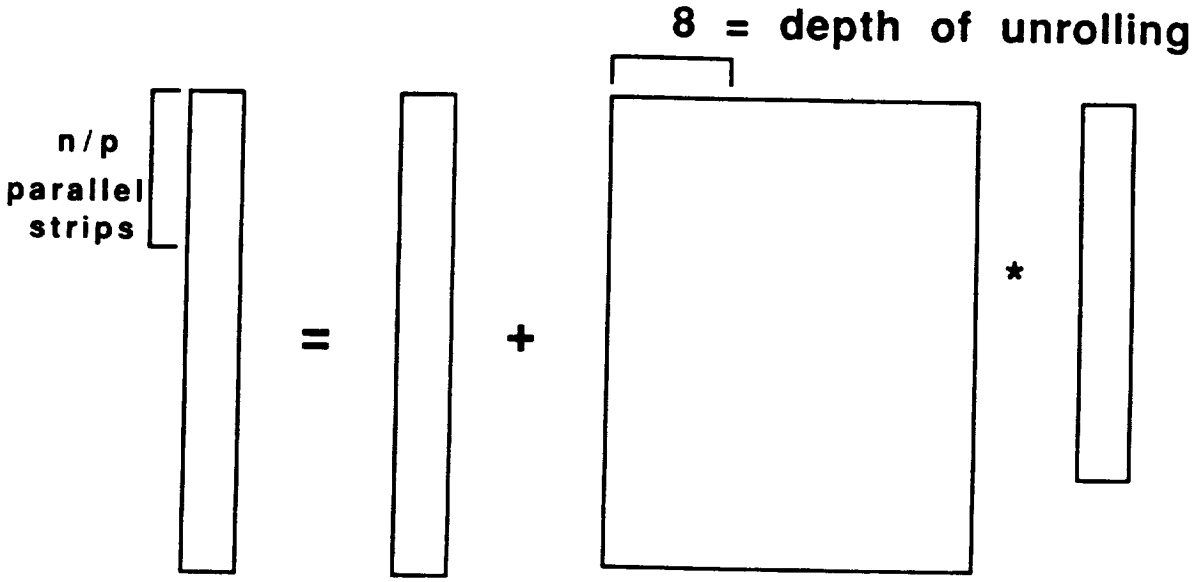

Figure 1: Autotasking of Supernodal Code.

of the authors (Phuong Vu) has obtained 188 MFLOPS single processor performance on a 8.5 nanosec Cray X-MP for BCSSTK33.

In our numerical tests on the Y-MP we have used some of the larger structural analysis matrices from [6]. These matrices are labeled BCSSTKxx and are described in more detail in the literature ([2, 6, 10]). In addition we have tested the performance on four structures matrices from the the Computational Mechanics Group at NASA Langley Research Center called NASAxxxx. These problems were provided by Olaf Storaasli. The largest problem in this group is a 54,870 degree of freedom model of the Space Shuttle Rocket Booster (NASASRB). Finally we created model problems using the 27 point operator and 5 unknowns per grid point on three dimensional cubic grids of size $N \times N \times N$. We considered the cases $N = 10, N = 15$, and $N = 17$, and the resulting matrices are labeled CUBE10, CUBE15, and CUBE17. These matrices can be considered as a model of the linear systems, which are solved in the fully implicit three-dimensional Navier-Stokes equations.

Unless noted otherwise, all timing results have been obtained in dedicated mode on the Cray Y-MP, Serial Number 1001, at Cray Research with a 6.49 nanosec cycle time. Several tests have been repeated on the Cray Y-MP at NASA Ames Research Center, Serial Number 1002, which has a cycle time of 6.3 nanosec. As to be expected, we observed that execution times scaled proportionally with the cycle time. Since the set of results obtained on the slower Y-MP is more complete, we report these results here for consistency. Based on these observations we expect a further performance improvement by about 8% on future Y-MP's, which will have a 6.0 nanosec cycle time.

The single processor factorization times are given in Table 1. The table contains the execution times and MFLOPS rates for three implementations of the key computational kernels. The highest performance was obtained with an assembly coded version of the key matrix multiplication routines (SGEMVS and SGMVIS). For comparison we list the results obtained with an all Fortran implementation of these subroutines with loop unrolling and tuned switch over from inline to subroutine calls (SGEMV8 and SGMVI8), and an in line Fortran version. Table 1 shows that the assembly coded version is more than twice as fast as the inline Fortan version. The highest single processor performance of 280 MFLOPS was measured on the model problem CUBE17, a sparse matrix of order 24,565 with about 1.5 million nonzeros in the unfactored matrix. On many of the large problems we obtain a one-processor performance in excess of 200 MFLOPS for the factorization, which

Table 1: **Single CPU Results for Supernodal Code**

Problem	Neqns	NumFact with SGEMVS+SGMVIS Time	NumFact with SGEMVS+SGMVIS MFLOPS	NumFact with SGEMV8+SGMVI8 Time	NumFact with SGEMV8+SGMVI8 MFLOPS	NumFact with Inline MV+MVI Time	NumFact with Inline MV+MVI MFLOPS
BCSSTK15	3948	0.91	182.81	1.10	150.45	1.90	87.03
BCSSTK16	4884	0.85	176.38	1.06	141.11	1.80	83.01
BCSSTK23	3134	0.67	177.11	0.79	149.72	1.29	92.63
BCSSTK24	3562	0.25	128.86	0.33	97.97	0.49	66.26
BCSSTK29	13992	2.34	168.54	2.86	137.69	4.64	85.07
BCSSTK30	28924	4.75	196.14	6.02	154.95	10.71	87.07
BCSSTK31	35588	11.00	232.38	13.22	193.31	24.42	104.69
BCSSTK32	44609	6.23	178.87	7.88	141.39	13.40	83.10
BCSSTK33	8738	5.05	238.86	6.05	199.11	11.15	108.21
NASA1824	1824	0.09	59.42	0.10	50.69	0.11	46.14
NASA2910	2910	0.21	90.10	0.27	80.14	0.35	61.10
NASA4704	4704	0.33	108.14	0.39	89.49	0.54	65.46
NASASRB	54870	20.21	231.71	24.99	187.45	48.08	104.64
CUBE10	5000	3.63	254.58	4.34	212.62	7.97	115.88
CUBE15	16875	57.99	278.79	67.75	238.60	131.21	123.21
CUBE17	24565	115.24	280.75	134.20	241.07	259.52	124.66

Table 2: **Multiple CPU Results for Supernodal Code (MFLOPS)**

		Number of Processors						
Problem	Neqns	1 Proc.	2 Proc.	4 Proc.	5 Proc.	6 Proc.	7 Proc.	8 Proc.
BCSSTK15	3948	182.81	233.36	278.34	288.70	293.41	295.96	296.55
BCSSTK16	4884	176.38	225.85	265.78	270.63	272.41	273.58	274.95
BCSSTK23	3134	177.11	227.31	271.12	280.68	286.79	291.54	291.30
BCSSTK24	3562	128.86	146.08	155.46	156.31	156.39	157.33	155.93
BCSSTK29	13992	168.54	208.53	238.12	244.10	247.25	248.83	248.72
BCSSTK30	28924	196.14	263.87	323.61	333.42	340.65	345.47	346.67
BCSSTK31	35588	232.38	370.13	523.55	563.57	597.64	621.17	637.15
BCSSTK32	44609	178.87	229.93	269.41	276.53	280.82	282.82	283.77
BCSSTK33	8738	238.86	381.52	548.33	588.90	621.74	644.78	663.06
NASA1824	1824	59.42	63.67	64.51	64.55	64.60	64.55	64.54
NASA2910	2910	90.10	108.04	112.43	112.70	112.75	112.81	112.27
NASA4704	4704	108.14	122.98	131.34	132.22	132.61	132.51	132.66
NASASRB	54870	231.71	355.79	491.45	523.33	548.90	563.97	578.02
CUBE10	5000	254.58	422.04	649.36	710.10	759.57	781.20	826.67
CUBE15	16875	278.79	537.08	965.47	1147.54	1306.92	1451.81	1569.55
CUBE17	24565	280.75	547.77	1005.92	1202.64	1382.94	1545.12	1681.92

is more than two thirds of the peak performance of the machine.

The factorization time for the shuttle problem with 54,870 degress of freedom took about 20 seconds. The shuttle problem is to be considered very large, even by todays structural engineering standards. For comparison, the widely used structural analysis package MSC/NASTRAN imposed up to recently an upper limit of 64,000 degrees of freedom on its users. The fact that a model at the upper limits of todays engineering computing can be solved in a time which permits interactive processing is by itself remarkable, and demonstrates the computational power of both the Y-MP and the algorithmic approach pursued here.

In Table 2 the results for the parallel implementation of the supernodal code are listed. The highest performance was again obtained for the problem CUBE17, where we measured 1682 MFLOPS using eight processors. This

problem also shows the highest speed-up with about a factor of six. For most of the other problems our speed-up is considerably less. For some of the smaller problems we see less than a factor of two speed-up even when using all eight processors. This is not a problem with the hardware, but with the lack of parallelizable work. In the simple loop level approach pursued here, we depend on the size of the matrices used in the matrix multiply routines for extracting parallizeable work, i.e. the number of rows in the matrix in Figure 1. In order to make autotasking worthwhile, we need a minimum number of rows (depending of the size of the supernode) for each of the supernodal updates. For example if the number of columns NCOL in the supernode is 20 we will not obtain any benefits using autotasking until the number of rows NROW is at least around 200, but for for NCOL = 50 the benfits of autotasking start at NROW = 50 etc. For general sparse matrices there is no guarantee that this minimum number of rows is exceeded in every supernodal update. This explains the flat speed-up curves for some of the problems.

5 Summary

We have demonstrated that the supernodal general sparse code and parallel processing on the Cray Y-MP form a powerful combination allowing sparse matrix factorization at extremely high speeds. However, our results are preliminary and offer several possibilities for improvement. On the hardware side all future Y-MP's will have a cycle-time of 6 nanosec. The faster clock will give an improvement of about 8% over the results reported here without any code modifications. Further finetuning of the autotasked routines may yield more improvements. Finally, we need to consider the case of small supernodes, where other approaches are needed to gain further speed-ups. In spite of their preliminary nature we believe, however, that our results are the best currently available.

References

[1] P. Amestoy and I. Duff. *Vectorization of a multiprocessor multifrontal code.* Technical Report TR 88/3, CERFACS, Toulouse, France, October

1988.

[2] C. Ashcraft, R. Grimes, J. Lewis, B. Peyton, and H. Simon. Recent progress in sparse matrix methods for large linear systems. *International Journal on Supercomputer Applications*, 1(4):10 - 30, 1987.

[3] C.C. Ashcraft. *A vector implementation of the multifrontal method for large sparse symmetric positive definite linear systems.* Technical Report ETA-TR-51, Boeing Computer Services, 1987.

[4] I. Duff and J. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM TOMS*, 9(3):302 - 325, 1983.

[5] I. Duff and J. Reid. The multifrontal solution of unsymmetric sets of linear equations. *SIAM J. Sci. Stat. Comput.*, 5:633 - 641, 1984.

[6] I. S. Duff, Roger G. Grimes, and John G. Lewis. *Sparse Matrix Test Problems.* Technical Report CSS 191, Harwell Laboratory, Didcot, Oxon, England, 1987.

[7] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.

[8] A. George, M. Heath, J. Liu, and E. Ng. *Solution of Sparse Positive Definite Systems on a Hypercube.* Technical Report ORNL/TM-10865, Oak Ridge National Laboratory, Oak Ridge, TN, October 1988.

[9] A. George and J. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice Hall, Englewood Cliffs, 1981.

[10] J.G. Lewis and H.D. Simon. The impact of hardware gather/scatter on sparse Gaussian elimination. *SIAM J. Sci. Stat. Comp.*, 9(2):304 - 311, 1988.

[11] H.D. Simon. *Modern Algorithms for Supercomputing: Sparse Matrix Algorithms.* Technical Report, IEEE Computer Society, November 1988. Tutorial Notes, Supercomputing '88.

January 17, 1989

To: D. Bailey
E. Barszcz
R. Fatoohi
T. Lasinski

From: H. Simon

Subject: Informal review of paper

Please review the enclosed manuscript "Sparse Matrix Factorization at 1.68 GFLOPS". I am planning to turn it into an RNR report and to submit it to SIAM J. Sci. Stat. Comp.
I have already submitted an earlier version to the "Gordon Bell Award Competition". Because of the pressing deadline, only David Bailey had an opportunity to review this earlier version.